INTRODUCTION TO JAVA

Computer programming involves using programming language (e.g., C or Java or C++ or Python) that gets translated into the CPU (central processing unit) using a compiler (a platform independent of the computer program that converts source code- .java file, in to bytecode - .class file) to input instructions that can control a computer.

Java was created in the beginning of 1990 by James Gosling as an object-orientated programming language for efficient use of home appliances, however, was unfamiliarised until the late 1990s given that it was evidently slower than the pre-existing C/C++. With internet being developed in the late 1990s/early 2000s, its position prevailed, leading up to this day, where it is a widely used by programmers. Java consists mainly of two platforms being, JAVA SE (standard edition) which is used for PG application development on the desktop, JAVA ME (Mobile Edition) which is developed for use in portable small devices.

- VARIABLES & DATA-TYPE –

Variable (변수): 변하는 수 (값을 담아두는 공간) - "자료 형 (int)의 xx 라는 변수를 선언하고, 이 변수에 초기값 α를 대입한 코드"

Variable declaration (변수 선언)

```
int   xx   =   α
```

Variable initialization (변수 초기화)

int: Data-type
xx: Name of variable
=: operator
α: Variable contents

Decimal numbers and **binary numbers:** x/y/z/r is either 0 or 1

Decimal number

$$x \cdot (2^3) + y \cdot (2^2) + z \cdot (2^1) + r * (2^0) = \underbrace{(x \cdot 1000) + (y \cdot 100) + (z \cdot 10) + (r \cdot 1)}_{\text{Binary number}}$$

0	0	$0 \cdot 2^0 = 0$
1	1	$1 \cdot 2^0 = 1$
5	101	$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$
9	1001	$1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9$
10	1010	$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 10$

Byte: The basic unit of information which consists of 8 adjacent binary digits (bits)

Primitive data types (기본 자료 형 정리 표)

	TYPE	DATA TYPE	DEFAUT SIZE	DEFAULT VALUE	RANGE $-2^{(\#\ of\ bits\ -1)} \sim (2^{(\#\ of\ bits\ -1)} - 1)$	SPECIFIER/NOTES
Numeric	Integer	byte	1 byte (8 bits)	0	-128 ~ 127 $-2^7 \sim (2^7 - 1)$	%d Example: int n1 = 33, n2=10; result = n1 + n2; System.out.printf("%d %c %d = %d\n", n1, '+', n2, result); *Output: 33 + 10 = 43*
		short	2 bytes (16 bits)	0	-32,768 ~ 32,767 $-2^{15} \sim (2^{15} - 1)$	
		int	4 bytes (32 bits)	0	-2,147,483,648 ~ 2,147,483,647 $-2^{32} \sim (2^{32} - 1)$	
		long	8 bytes (64 bits)	0L	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 $-2^{64} \sim (2^{64} - 1)$	
	Character	char	2 bytes (16 bits)	\u0000	0~65,535 **a = 97**, b =98, c=99 **A = 65**, B = 66, C =67 Char a = "A" → A Int a → 97	%c
	Floating-point	float	4 bytes (32 bits)	0.0f	-3.4E38 ~ +3.4E38	%f (Number of decimal places by default is always 6 decimal places: 8 characters)
		double	8 bytes	0.0d	1.7E308 ~ + 1.7E308	

			(64 bits)			Example: double d1 = 10; System.***out***.printf("d is%5.1f", d1); *Output: d is 10.0* *in the '%5.1f', 5 indicates the width and 1 represents the number of decimal places
Boolean		boolean	1 byte (8 bits)	false	True/false	%b

Literal: Any constant value which can be assigned to the variable is called literal/constant.
Constant character (escape): Any constant value which can be assigned to the variable is called literal/constant.

ESCAPE CHARACTERS **(must be all in quotation marks)**	**DESCRIPTION**	**EXAMPLE INPUT**	**EXAMPLE OUTPUT**
\t	It is used to insert a **tab** in the text at this point.	String str = "Andrew\tGarfield" *System.out.println(str);*	Output: Andrew Garfield
\'	It is used to insert a **single quote** character in the text at this point.	String str2 = "Wall Street\'s"; *System.out.println(str2);*	Wall Street's
\"	It is used to insert a **double quote** character in the text at this point.	String str3 = "\"JavaTpoint\""; System.out.println(str3);	"JavaTpoint"
\\	It is used to insert a **backslash character** in the text at this point.	*String str4 = "And\\Or";* *System.out.println(str4);*	And\or
\n	It is used to insert a **new line** in the text at this point.	String str5 = "the best way\nto communicate \nan idea \nis to act it out"; *System.out.println(str5);*	the best way to communicate an idea is to act it out

Casting: When you assign a value of one primitive data type to another type (when size of the data type is different).

byte → short → char → int → long → float → double

Widening Casting (automatically): Converting a smaller byte value to a larger byte space.	**Narrowing Casting (manually):** Converting a larger byte value to a smaller byte space (must indicate using parenthesis)
int number = 10; long number2 = number; *Since, long > int, this is an example of widening casting	double pi = 3.14; int pi2 = (int)pi; *Since, long < int, this is an example of narrowing casting
Output: 10	*Output:3*

- OPERATOR -

Operator: Special symbols that perform specific operations on one, two, or three operands, and then return a result.

<table>
<tr><th>Precedence</th><th colspan="2">Type</th><th>Operator</th><th>Description</th><th>Associativity</th></tr>
<tr><td>1</td><td colspan="2">First Operator</td><td>( )</td><td>Function call</td><td>→</td></tr>
<tr><td rowspan="3">2</td><td colspan="2" rowspan="3">Unary Operator</td><td>!</td><td>Logical negation (opposite)</td><td rowspan="3">←</td></tr>
<tr><td>++ x</td><td>Prefix increment: increments the value of x, and then returns the incremented value.

x = 1
y = ++x;
output: x is 2, y is 2</td></tr>
<tr><td>x++</td><td>Postfix increment:
increments the value of x, but returns the original value that x held before being incremented.

x = 1
y = x++;
output: x is 2, y is 1</td></tr>
<tr><td></td><td colspan="2"></td><td>--</td><td>Postfix/prefix decrement
(Works the same as the increments)</td><td></td></tr>
<tr><td rowspan="3">3</td><td rowspan="15">이항연산자</td><td rowspan="3">Multiplicative Operator
(산술연산)</td><td>*</td><td>Multiplication</td><td rowspan="15">→</td></tr>
<tr><td>/</td><td>Division</td></tr>
<tr><td>%</td><td>Modulo: Returns the remainder of the two numbers after division</td></tr>
<tr><td rowspan="2">4</td><td rowspan="2">Additive Operator
(산술연산)</td><td>+</td><td>Addition</td></tr>
<tr><td>-</td><td>Subtraction</td></tr>
<tr><td rowspan="6">5</td><td rowspan="6">Relational Operator
(비교연산)</td><td><</td><td>Less than</td></tr>
<tr><td><=</td><td>Less than (inclusive)</td></tr>
<tr><td>></td><td>Greater than</td></tr>
<tr><td>>=</td><td>Greater than (inclusive)</td></tr>
<tr><td>==</td><td>Equality</td></tr>
<tr><td>!=</td><td>Inequality</td></tr>
<tr><td>6</td><td>Bitwise AND
(비트연산)</td><td>&</td><td>Binary AND Operator copies a bit to the result if it exists in both operands.

a = 0011 1100
b = 0000 1101
(A & B) will give 12 which is 0000 1100</td></tr>
<tr><td>7</td><td>Bitwise OR
(비트연산)</td><td>|</td><td>Binary OR Operator copies a bit if it exists in either operand.

(A | B) will give 61 which is 0011 1101</td></tr>
<tr><td rowspan="2">8</td><td>Logical AND
(논리연산)</td><td>&&</td><td>expression1 && expression2

true only if both expression1 and expression2 are true</td></tr>
<tr><td>Logical XOR</td><td>^</td><td>If only one of the expressions are true
int a =10;
int b = 5;

a >b ^ a = 10 → false
a >b ^ a = b → true</td></tr>
</table>

		Logical NOT	!	The opposite value int a =10; int b = 5; (! (a > b)); → false (! (a < b)); → true	
9		Logical OR (논리연산)	||	expression1 || expression2 *true if either expression1 or expression2 is true*	
10	Conditional Operator (Ternary Operator)		? :	variable = **Expression ? expression1 : expression2** If the Expression is true, expression1 is assigned to the variable. If the Expression is false, expression2 is assigned to the variable.	
11	Assignment Operator (대입연산)		=	a = b → a = b;	←
			+=	a += b → a = a+ b;	
			-=	a -= b → a = a - b;	
			*=	a *= b → a = a * b;	
			/=	a /= b → a = a / b;	
			%=	a %= b → a = a % b;	

- CONTROL STATEMENTS –

Statement (문)	Executable statement (실행문)	Sequential statement (순차문)		
		Control statement (제어문)	IF (조건문)	if, switch
			LOOP (반복문)	for, while,do~while
	Non-executable statement (비실행문)	Annotation (// /* */)		

1. IF STATEMENTS:

Description:	Example:
if (*condition) {* **실행문;** ***} else if (condition2) {*** **실행문 2;** **} else {** **실행문 3;** **}**	**public static void main(String[] args) {** **int seoulLunchPrice = 4000;** **if(seoulLunchPrice>=7000) {** **System.*out*.println("It's expensive");** **}else if(seoulLunchPrice>=6000){** **System.*out*.println("Wish it was cheaper");** **}else if(seoulLunchPrice>=5000){** **System.*out*.println("Perfect");** **}else {** **System.*out*.println("It's too cheap");**

2. SWITCH STATEMENTS:

Switch statements select execution statement according to the value of the variable

- When programmer sets a variable, the computer compares the variable with the value of each case.
- When the two equalize, the program runs that case's command (executable statement).
- If there is no case with the same value, it runs the default command

*The program runs until it meets a break, in which case, it escapes the whole switch command.

```
Int/String variable = 숫자/문자

Switch (variable) {
        case 값 1:
        실행문 1;
        break;

        case 값 2:
        실행문 2;
        break;

        default:
        실행문 3;
                        }
```

```
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
        System.out.print("insert score: “);

int hak = sc.nextInt();
int hak = sc.nextInt();
int temp = (hak==100)?hak-1:hak;

switch(temp/10){
        case 9:
        System.out.println("A ");
        break;
        case 8:
        System.out.println("B");
        break;
        case 7:
        System.out.println("C ");
        break;
        case 6:
        System.out.println("D ");
        break;
        case 5: case 4: case 3:
        case 2: case 1: case 0:
        System.out.println("F ");
        break;

        default:
        System.out.println(“Not a valid number");
                }
        }
```

3. FOR STATEMENTS:

- initial value: the variable is given a value (either int or string)
- condition: the condition is run and in the case it is satisfied for that value, it executes the command (실행문).
- Whether the condition is satisfied or not, the value of the variable is changed by the increase/decreased, indicated by the last

```
For (initial value ; condition ; change) {
실행문;


//E.g – continue = skip that value but continue for the rest
For (int i=9 ; i<10 ; i++) {
if (i==5) {
continue ;  }}
Output: 0 1 2 3 4 6 7 8 9

//E.g – break = break out of the cycle
For (int i=9 ; i<10 ; i++) {
if (i==5) {
break ;
}
}

Output: 0 1 2
```

4. WHILE STATEMENTS:

The number of repetitions is decided by the condition and only when the condition is satisfied, the command (within the block) is executed.

```
While (condition) {
        실행문;
        }
```

4.1 DO ~ WHILE STATEMENTS:

The condition is compulsorily executed once and only if the condition is true, it repeats.	Do { 실행문;) while (condition);

5. TERNARY OPERATOR:

The ternary operator consists of a condition that evaluates to either true or false, plus a value that is returned if the condition is true and another value that is returned if the condition is false.	String/int variable = (condition)? "x": (condition 2)? "y": "z"; //x: what is puts in the variable if condition 1 is true //y: what is puts in the variable if condition 2 is true //z: what is puts in the variable if neither conditions are true. Example: public static void main(String [] args) { int age = 21; System.out.println("Age is: " + age); String msg = (age<0) ? "not a valid number": (age >=18) ? "adult" : "child"; System.out.println(msg); }

- ARRANGEMENTS–

Arrangements: A set of the same data type where multiple data can be used with one name. A collection of several variables.

- The size of arrangement is set in the beginning and cannot be changed

```
int/string [] variable = new int/string []
```

Arrangement declaration (배열 선언)

Length
Index order = length -1

1. INSERTING VALUES IN THE ARRAY:

1.1 Integer:

```
int [] Arr = new int[5];
for (int i = 0 ; i < Arr.length ; i++){
        Arr[i] = i+1;
    System.out.println(Arr[i]); }
```

output:
1
2
3
4
5

1.2 String

```
String [] Arr2 = {"A", "B", "C"};
for (int i = 0 ; i < Arr2.length ; i++){
    System.out.println(Arr2[i]);}
```

2. MULTIDIMENTIONAL ARRAY

열 (Row) 행 (Column) (0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2,0) (2,1) (2,2) 3 x 3 의 행렬	int [][] matrix = new int [3][3]; matrix [0][0] = 1; matrix [0][1] = 2; matrix [0][2] = 3; matrix [1][0] = 4; matrix [1][1] = 5; matrix [1][2] = 6; matrix [2][0] = 7; matrix [2][1] = 8; matrix [2][2] = 9; OR int [][] matrix = {{1,2,3}, {4,5,6}, {7,8,9}};	열 (Row) 행 (Column) 1 2 3 4 5 6 7 8 9 3 x 3 의 행렬

3. COPYING ARRAY

Method 1:	Method 2:	Method 3:

<table>
<tr><td>Copying array ‘score’ to a new variable named ‘s’ using the for command</td><td>Copying array ‘score’ to a new variable named ‘s’ using the array copy class.

In System.arraycopy:
- score: original variable
- 0: the initializing index value
- s: copied variable
- 0: from which index (in s) will it insert the copied values
- score.length: the length of the array that is to be copied.</td><td>Copying array ‘score’ to a new variable named ‘s’ using the Copyof() method from the class Arrays.</td></tr>
<tr><td><pre>int[] score = {100,10,20,30,40};
int[] s = new int[score.length];

for(int idx=0 ; idx<score.length ;
idx++) {
s[idx] = score[idx];
}
System.out.printf("score[%d]=%d \t
s[%d]=%d\n", idx, score[idx], idx,
s[idx]);
}</pre></td><td><pre>int[] score = {100,10,20,30,40};
int[] s = new int[score.length];

System.arraycopy(score, 0, s, 0,
score.length);

for(int idx=0 ; idx<score.length ;
idx++) {
System.out.printf("score[%d]=%d \t
s[%d]=%d\n", idx, score[idx], idx,
s[idx]);
}</pre></td><td><pre>int[] score = {100,10,20,30,40};
int[] s = null;

s = Arrays.copyOf(score, score.length);
for(int idx=0 ; idx<s.length ; idx++) {
System.out.printf("score[%d]=%d \t
s[%d]=%d\n",

idx, score[idx],idx,s[idx]);
}</pre></td></tr>
<tr><td colspan="3">Output:
score [0] = 100 s[0] = 100
score [1] = 10 s[1] = 10
score [2] = 20 s[2] = 20
score [3] = 30 s[3] = 30
score [4] = 40 s[4] = 40</td></tr>
</table>

4. IMPROVED FOR COMMAND

- Allows the program to read the command but not change in (increment or decrement it). Starting from the value in the first array, each array value gets printed in a variable defined by the programmer.
- Example 1: 65 is inputted in the variable “number” and this is printed, then 235 is inputted in the variable “number” and this is printed…etc…)
- For the improved for command, you cannot change a value in the array, whereas with the normal for command, you can. (See example 2)

<table>
<tr><td>Example 1:</td><td>Example 2:</td></tr>
<tr><td><pre>int[] array = {65,235,32,58,92};
for (int number : array) {
System.out.print(number + "\t");</pre></td><td><pre>int[] arr = new int[3];
for(int i=0 ; i<arr.length ; i++) {
arr[i] = 5;}

for(int temp : arr) {
temp = 9;}

for(int temp : arr) {
System.out.println(temp);</pre></td></tr>
<tr><td>Output:
65 235 32 58 92</td><td>Output:
5
5
5</td></tr>
</table>

5. REFERENCE TYPE

Primitive Type: Store literal value of an integer, a real number, or a literal data type

```
String name1 = "Amy";
```

Reference Type: Does not store literal value but stores the address of where the literal value is stored.

```
String name1 = new String("Amy");
```

- When a reference type variable is created, the variable will have a null value
- When you make a variable with the same address **`(int[] variable1 = variable2;)`** and change something in either variable, the other variable also changes value in the same way, as they share the same address. (if an array is copied, they do not share the same address).

```
int [] arr1 = {1,2,3}
int [] arr3 = arr1;
arr3[0] = 4;
System.out.print(arr3[0] + "\t");
System.out.print(arr1[0]);
```

Output: 4 4

Comparing memory address:	Comparing the actual String value:
`String name1 = new String("Amy");` `String name2 = new String("Amy");` `System.out.println(name1 == name2);`	`String name1 = new String("Amy");` `String name2 = new String("Amy");` `System.out.println(name1.equals(name2));`
false	*true*

- OBJECT ORIENTATED PROGRAMMING (객체지향) –

A programming paradigm that relies on the concept of classes and objects.

4 fundamental concepts of OOP:

1. **Inheritance (상속):** When the child class inherits the properties and functions of the adult class, avoiding the need to copy the same codes. Saves maintenance and developing time.
2. **Encapsulation (캡슐화):** A mechanism of concealing the data (variables) as a single unit which are made accessible according to permissions (access modifier).
3. **Polymorphism (다형성):** The ability of an object to take many forms and therefore perform the same action in many different ways.
4. **Data abstraction (추상화):** The process of reducing the object to its essence so that only the necessary characteristics are exposed to the users. Abstraction defines an object in terms of its properties (attributes), behaviour (methods), and interfaces (means of communicating with other objects).

```
Public class MemberMain {

        public static void main(String[] args) {

                Member m = new Member();
                Member m2 = new Member();

                if(m == m2) {
                        System.out.println("Object m and m2 are the same");
                } else {
                        System.out.println("Object m and m2 are different");
                }
        }
}
```

The MemberMain class has created a object class named "Member", using the main() method.

The main() method is the starting point to run the program. Therefore there is one class with the main() method and all the other classes are made to be used in other classes

Creating a variables and m2 that has the type "Member"

Class MemberMain has been created to run the program where as Class Member has been created to be utilized in other classes.

Output: Object m and m2 are different

Although variables m and m2 have the same data type as objective variables, they have individualized addresses

Package: It is a concept used to bind classes that can functionally affect multiple classes in programming, and to effectively call them within the access range. Java(JDK) have already constructed classes (approx. 3000) and it can be found here:

https://docs.oracle.com/javase/8/docs/api/index.html

- Importing: If you want to create a package and then use classes from different packages after you create a class, you must use the import keyword.
  - ✓ Classes in the same package do not require import.

Object: Consists of data and method of the same type

1. CLASS

The process of using a ***class*** (the plan - 설계도) for the ***object*** that is created using this class, (creation – 피조물) is called a ***instance*** (클래스 이용해 객체 생성).

- Just as two cars of the same model, share the same plans and parts but is still two 'difference cars', two objects (in java) that share the same class are considered two difference objects. A new class is made using:

```
ClassName variableName = new ClassName();
```

1.1 FIELD (필드): Place where the object's characteristics or specific values are saved. It is the variable within the class.

- A class variable is defined within the block of the class by using the word 'static' in front of the variable's type. (This variable is shared with the entire object).

```
ClassName.ClassVariableName
```

- An instance variable is also defined within the block of the class but the word static is not added. Since each object has its own address, each object has its own value for this variable

```
ObjectName.InstanceVairableName
```

EXAMPLE	
Class 'Andante' under main class	`Class Avante {` `String color;` `Static String company = "Hyundai";` `}`

Main class

```
Public class VarEx {                                   VarEx: File name

        public static void main(String[] args) {       The main() method is what runs the program and this method is within the class named VarEx

                System.out.println(Avante company : ” + Avante.company);   Avante company: Hyundai
                                                       Avante: Class Name
                                                       Field 2: company
                                                       (Class variable)
                Avante a1 = new Avante();
                Avante a2 = new Avante();

In each object, the instance variable ‘color’ has been given a value of ‘white’ for a1 and ‘black’ for a2
                a1.color = “white”;    Field 1: color (Instance variable)
                a2.color = “black”;

Printing an instance variable
                System.out.println(“a1 color: ” + a1.color);
                System.out.println(“a2 color: ” + a2.color);        Result:
                                                                    a1 color : white
                                                                    a2 color : black
Although this is a class variable, it is used like an instance variable (as class variables are shared with the entire object
                System.out.println(“a1 company: ” + a1.company);
                System.out.println(“a2 company : ” + a2.company);   Result:
                                                                    a1 company : Hyundai
                                                                    a2 company : Hyundai
As company is a class variable, shared with both objects, by changing company a1 to Kia, a2 also changes (as a1 and a2 are within the “company” class variable
                a1.company = “Kia”;

                System.out.println(“Avante company: ” + Avante.company);
                System.out.println(“a1.company: ” + a1.company);
                System.out.println(“a2.company: ” + a2.company);    Result:
                                                                    Avante company: Kia
                                                                    a1 company : Kia
                                                                    a2 company : Kia
```

1.2 CONSTRUCTOR (생성자): It is used when making a new object. Although it is like a method, for a constructor, its name and the class name need to be equivalent, and it **does not** have a return value

```
Member member = new Member();
```
Member (), that is added after the ‘new’ operator is a constructor

EXAMPLE	
Method class	`Public class Student {` `String name;` `int grade;` `String department;` — 3 fields: name, grade departmnet `Student(String n, int g, String d) {` `Name = n;` `Grade = g;` `Department = d;` `}` — Constructor that receives an input of 3 object variables `Student(String n, int g) {` `Name = n;` `Grade = g;` `}` — Constructor that receives an input of 2 object variables `}`
Main class	`Public class StudentMain {` `public static void main(String[] args) {` `Student stu = new Student (“Anna”, 4, ”Economics”);` — Constructing an object called “Student” `System.out.println(stu.name);` `System.out.println(stu.grade);` `System.out.println(stu.department);`
Result	Anna 4 Economics

1.3 METHOD (메서드): A logic that is created separately. Iif given the appropriate data it will return the result (It is an action that an object is able to perform).

- If the method has a return value, after the executable command, must return a value.
- If method ends with return without a return value, it is equivalent to ending the method.
- If the method has no return value, before the method name, must write void (**`void MethodName()`**).
- One can make a method that does not expect a specific number of values (add …):

```
Void methodName(int … x) {
        int sum = 0;
```

```
For (int i=0 ; i<x.length ; i++) {
Sum == x[i];
```

- If the method starts with 'static', use this method without making an object:

EXAMPLE (with static)	
Method class	`Static void printA() { System.out.println("A")`
Main class	`Method.printName()`
Result	A

EXAMPLE2 (without static)	
Method class	`void printB() { System.out.println("B")`
Main class	`Method m = new Method(); m.printB();`
Result	B

Overloading: When there are multiple methods of the same name **within the same class** with at least one of the following different: method variable's type, its count or order.

Overriding: Saving multiple methods of the same name **in another class**.

```
Class Operator {
int multiply(intx, int y) {
Return x * y;
}
Double multiply (double x, double y) {
Return x * y;
}
Double multiply (int x, double y) {
Return x * y;
```

Printing 4 types of the "multiply" method that have different object variable types

this: Keyword used when using an operator of the same class. It is also used as an object of a reference variable.

- "this" is used to distinguish a declared variable with an instance variable and save the instance variable, when they share the same name, within the same method
- by adding a "this", the program looks for the variable made in its current class, and then if not found, it looks for the variable in the parent class.

```
Public class Car {

        String name;
        int company;
        String type;

        Car () {
                this("white", "Kia", "SUV");
        }

        Car(String color, String company, String type) {
                this.color = color;
                this.company = company;
                this.type = type;

        Car(String com, String t)
                this("white", com, t);
        }

        Car(String t) {
                this("white, "Kia, t);
        }

        public String toString() {
                return color + "-" + company + "-" = type;
        }
}
```

3 fields: name, company, type

Constructor 1:

Constructor 2: Creates its own objects: variable color, variable company, variable type.

'this' is used as the parameter name and the field's name is the same.

Constructor 3

Constructor 4

Access modifier: Can be specified for Class, Method/constructor and variables	**Access Modifier Range**			
	All the Classes	**Inheritance Relationship**	**Same Package**	**Same Class**
Public	○	○	○	○
Protected	x	o (If the parent class use Protected, then only the child class can use it).	○	○
(Default) – If a access modifier is not stated, then the program reads it as default	x	x	○	○
private	x	x	x	○

final: The variables final value in which after, it cannot be changed.

- final class: inheritance is not possible (a string class and a Math class
- final method: an overriding method where alterations are not possible,
- final variable: value of the variable cannot be changed
- final static: used as "**`static final type variableNames".`** (To define something like PI or the speed of a specific vehicle

- INHERITANCE (상속) –

Inheritance: Mechanism in which a child object (class) acquires all the properties and behaviours (methods and fields) of a parent object (class).

- A subclass can have only **one superclass.** This is because Java does not support multiple inheritances with classes. Although with interfaces, multiple inheritances are supported by java.
- A subclass inherits all the members **(fields, methods, and nested classes)** from its superclass. Constructors are not members, so they are not inherited by subclasses.
- **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods (like getters and setters) for accessing its private fields, these can also be used by the subclass.

```
Class ChildClass extends ParentClass {
```

1.1 SUPER

Super: Reference variable that is used in the Child's class to identify objects of the parent class.

- [super.]: A reference variable that is used to refer parent class **objects**. It calls parent class' variables and methods.
- [super()]: A reference variable that is used to refer parent class constructors.

```
Class Parent {
       int number = 3;

Parent() {
       System.out.println("Parent object");
```

```
Class Child Extends Parent {
       int number = 2;
       Child() {
              System.out.println(Child object");
       }

       Void print () {
```

```
                                        int number = 1;
                                                System.out.println(number);
                                                System.out.println(this.number);
                                                System.out.println(super.number);
```

```
Public class SuperEx {

      Public static void main(String[] args) {

  Child child = new Child();
  Child.print();
        }
}
```

Output
Parent object
Child object
1 (looks for closest value of variable 'number' which is the one within in the method)
2 (printing the instance variable of the object 'child' in the 'Child' class – 2)
3 (looks for value of variable 'number' the parent object).

- OTHER TIPS–

NAME	DESCRIPTION	EXAMPLE
Scanner class:	Scanner is a class in java.util package, used for obtaining user input of the primitive types like int, double, etc. and strings.	***package com.lec.ex;*** ***import java.util.Scanner;*** ***public class Example*** { public static void main(String[] args) { Scanner sc = new Scanner(System.in); //String input System.out.println("Name: "); String name = sc.nextLine(); // Character input ((charAt(0) returns first character in that string) System.out.println("Gender: "); char gender = sc.next().charAt(0); // Numerical data input System.out.println("Age: "); int age = sc.nextInt(); // Others (byte, short and float can also be used System.out.println("Mobile num: "); long mobileNo = sc.nextLong(); System.out.println("Height: "); double height = sc.nextDouble(); System.out.println("The user "+ name + " (" + gender + ") " + "is " + age + " years old" + " with the phone number " + mobileNo + " and height of " + height); sc.close(); } }
Math.random (Class Math)	Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned	public static double random() int computer = (int) (Math.random() * 3); // 0,1,2

	values are chosen pseudo-randomly with (approximately) uniform distribution from that range.	`// (int) is used in order to convert a double value (random value that the computer generated), in to a integer = Manual Narrowing Casting`
Importing classes	Importing all the classes from the relevant package	**`Import PackageName.*`**
Instance of	Checking which class the object is from, or inherited from	**`Object instanceOf type`** **`Example:`** **`Public class Banana implements fruit {`** **`}`** **`Public apple {`** **`}`** **`Main class:`** **`Banana fruit = new Banana ();`** **`System.out.println(fruit instancof Banana);`** **`System.out.println(fruit instancof Apple);`** `Result:` `True` `True`